

# C868 – Software Capstone Application Design

## Task 2 – Section C



Capstone Proposal Project Name: Ashlar CRM

Student Name: Daniel Berry

# Table of Contents

<b><i>Application Design.....</i></b>	<b><i>4</i></b>
Summary .....	4
Entity Relationship Diagram (ERD) .....	4
Class Diagram .....	6
UI Design .....	7
<b><i>Unit Test Plan .....</i></b>	<b><i>8</i></b>
Introduction.....	8
Purpose .....	8
Overview .....	8
Unit Test Plan .....	9
Items .....	9
Features .....	9
Deliverables .....	9
Tasks .....	9
Needs .....	9
Pass/Fail Criteria .....	10
Specifications .....	10
Procedures .....	11
Results.....	11
<b><i>Artifacts.....</i></b>	<b><i>11</i></b>
Source Code.....	11
Link to Live Version .....	11
<b><i>Application Maintenance Guide.....</i></b>	<b><i>13</i></b>

<b>Prerequisites.....</b>	<b>13</b>
<b>Installing Docker Desktop .....</b>	<b>13</b>
Installing Docker Desktop on macOS .....	13
Installing on Windows.....	13
Installing on Linux .....	14
<b>Using Sail to Start Your Development Environment .....</b>	<b>14</b>
Using the Zip Archive: .....	14
Using Git and Composer .....	15
<b>Having Issues Installing the Development Environment? .....</b>	<b>15</b>

# Application Design

## Summary

This section includes the design documents used during the creation of the software. I have included the class diagram, the entity diagram, low-fidelity wireframes, and prototype designs. We use these artifacts to document and validate adherence to established requirements. The development team then uses each document to begin implementation and development. These documents provide a structured and precise framework for developing the software.

## Entity Relationship Diagram (ERD)

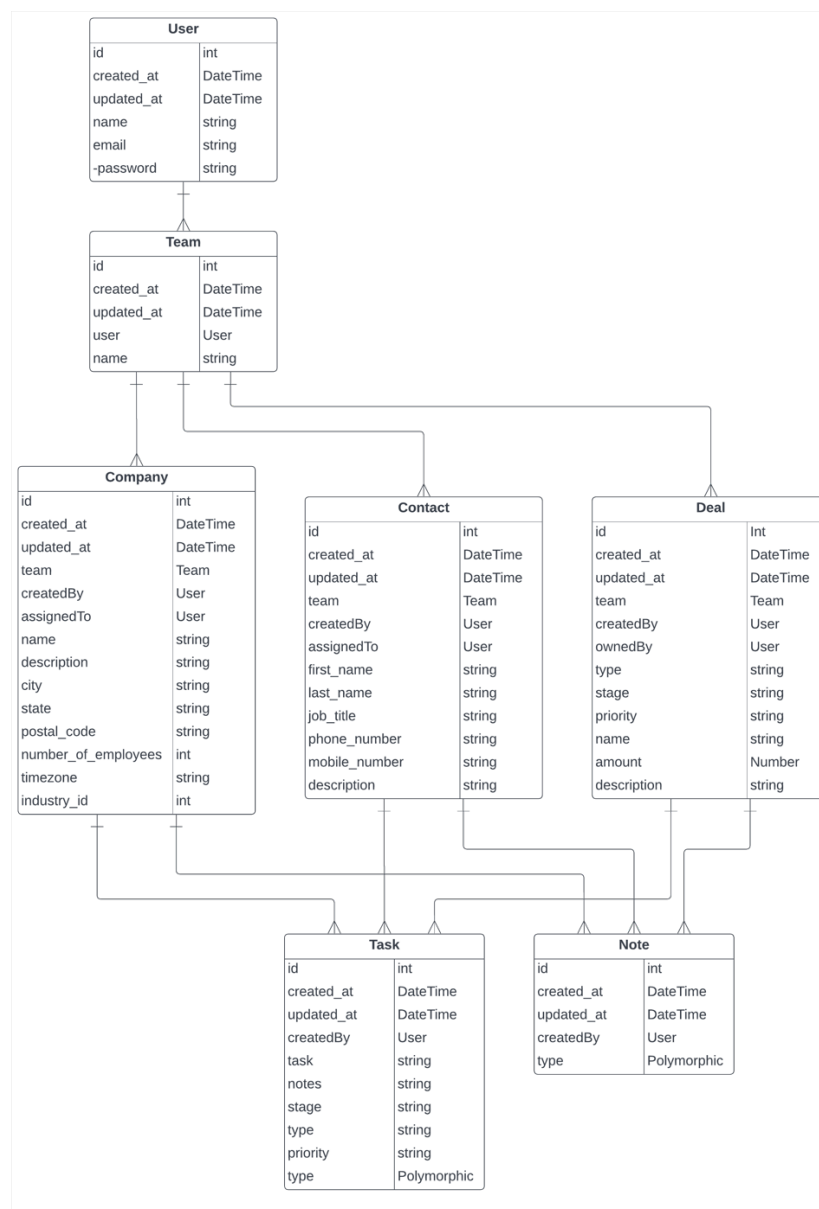
The Entity Relationship Diagram, or ERD, shows the relationships of the different entities stored in our database. The diagram will show each entity's attributes and the relationships between them. This diagram serves as a schema overview for database development.



## Class Diagram

The class diagram is a type of unified modeling language (UML) structure diagram that allows the development team to describe the application's structure. Due to size of the application, we have only included the base models that represent the entities described in our ERD. This application uses Eloquent ORM, all attributes are inherited automatically.

The class diagram below illustrates these models by showing their attributes, methods, and relationship to other classes.



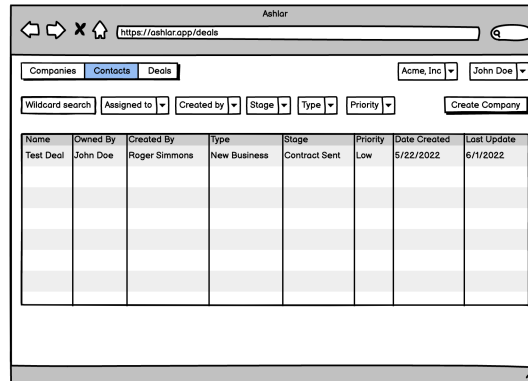
## UI Design

During the Inception phase of our development cycle, the design team created a set of wireframes that served as a blueprint for the application. Doing the wireframes on the front end of the project before starting development ensured that the application was structured correctly and would result in an easy-to-use end product. Wireframes also helped the team validate the proper understanding of the design requirements and ensure we met the stakeholder's needs.

The wireframes below depict the major screens used throughout the application, the authentication pages and the companies, contacts, and deals dashboards.

The wireframes show the following screens:

- Login:** A simple form with fields for Email Address and Password, a Login button, and a link for "Need an account? | Forgot Password".
- Register:** A form with fields for Team Name, Your Name, Email Address, and Password (repeated), a Register button, and a link for "Already have an account? | Register".
- Dashboard:** A welcome message "Welcome to Ashlar" with a navigation bar (Companies, Contacts, Deals) and a sidebar with links to Companies, Contacts, Deals, and Profile.
- Forgot Password:** A form with an Email Address field and a Send button.
- Companies:** A table with columns: Name, Assigned To, Created By, City, State, Industry, Date Created, and Last Updated. The table contains one row for "Acme Inc" and several empty rows.
- Contacts:** A table with columns: First N, Last N, Assigned T, Create, Email, Job Title, Phone Nu, Mobile N, Date Creat, and Last Updat. The table contains one row for "Lori Harbor" and several empty rows.



# Unit Test Plan

## Introduction

Unit tests validate small pieces of our application, such as methods or attributes in a class. It usually deals with testing fine details at a low level only. In our case, we will be testing some methods on our model classes, model connections, and relationships between different parts of an app.

In this section, we will describe a specific unit test that ensures the relevant table models are not missing and have the correct and expected names. The example below is for the Deal model, but every model begins with this test.

## Purpose

Since this is a database-driven application, ensuring that each table has the correct column names is extremely important.

## Overview

We use Laravel's schema and migration system to create tables as defined in our ERD. While creating our migrations, a column name could potentially be left out or misspelled, which would cause the application to fail.



## Unit Test Plan

### Items

To perform this test, we used Laravel's integrated PHPUnit test suite. Our docker contains a separate test database that can be freshly migrated each time we run our suite of tests. All tests are stored within the /tests folder under the respective subfolder. In Laravel, you create a test by running an artisan command. In this case we will run `php artisan make:test UserTest --unit`. This creates a new file named `DealTest` within the tests/unit folder that extends the application's base `TestCase` class which sets up our testing tools.

### Features

Because unit tests focus on small specific portions of our application, in this case a model's database schema, it is important that we run the test in isolation from larger feature sets.

### Deliverables

We can run the test from the IDE, PHPStorm by JetBrains, or the command line. For the sake of this document, we will describe running the test from the command line. Running the following command will produce a successful result from the application root if all information contained in the migration schema is correct.

### Tasks

To run the described test, ensure that docker is running on your machine, and start the development environment by running `sail -d` from your command line. Once the development environment is running, navigate to the root folder using your command line and run `sail test --filter DealTest` as shown in the figure below.

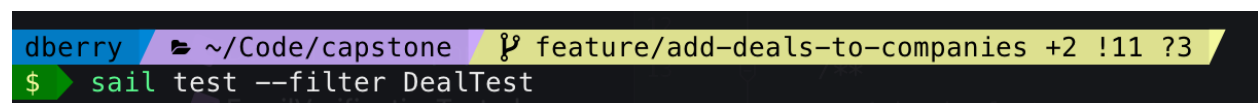
A terminal window screenshot with a dark background. The top bar shows the user 'dberry', the current directory '~/Code/capstone', and the file 'feature/add-deals-to-companies +2 !11 ?3'. The command prompt '\$' is followed by the command 'sail test --filter DealTest'.

Figure 1 Command line code to run the DealTest

### Needs

- Docker installed on your machine. You can learn more about installing Docker by visiting <https://docs.docker.com/get-docker/>

- The application code and project files.

## Pass/Fail Criteria

The test is considered successful if the test passes with a green checkmark and shows the word "PASS" in green. If a test fails, the QA tester or developer should look at the corresponding migration file to ensure all of the required columns are present and spelled correctly. A failing test should be logged and reported to the development team.

## Specifications

Below is a screenshot of the unit test described above. This test is in the /tests/Unit folder of the application.

```
1  <?php
2
3  namespace Tests\Unit;
4
5  use Illuminate\Foundation\Testing\RefreshDatabase;
6  use Schema;
7  use Tests\TestCase;
8
9  class DealTest extends TestCase
10 {
11     use RefreshDatabase;
12
13     /**
14      * A basic feature test example.
15      *
16      * @return void
17      */
18     public function test_deals_database_table_has_expected_columns()
19     {
20         $this->assertTrue(
21             Schema::hasColumns( table: 'deals', [
22                 'team_id',
23                 'created_by_id',
24                 'owned_by_id',
25                 'type',
26                 'stage',
27                 'priority',
28                 'name',
29                 'amount',
30                 'close_date',
31             ])
32         );
33     }
34 }
```

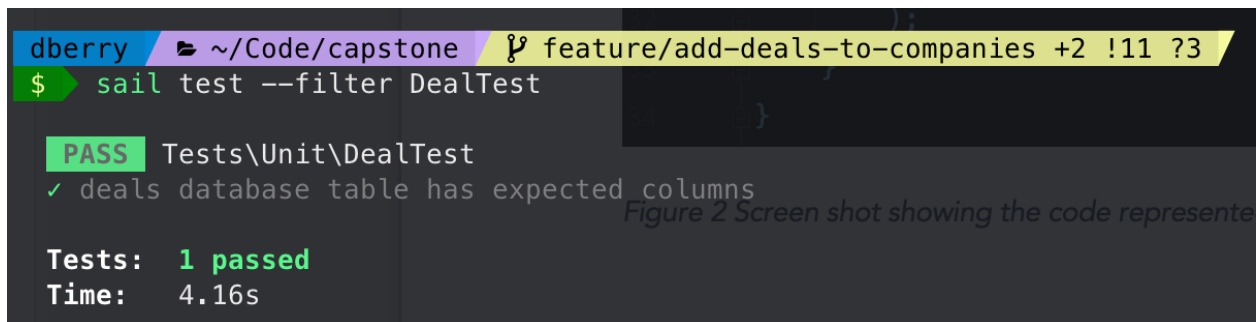
Figure 2 Screenshot showing the code represented in our described test.

## Procedures

To execute this unit test I opened the project inside the Docker development environment. Once the development environment was fully loaded, I opened the command line and ran `sail test -filter DealTest`. The results of the test were displayed in the console once the test finished running.

## Results

Below is a screenshot showing the results of running this unit test.



```
dberry ~/Code/capstone feature/add-deals-to-companies +2 !11 ?3
$ sail test --filter DealTest

PASS Tests\Unit\DealTest
✓ deals database table has expected columns

Tests: 1 passed
Time: 4.16s
```

Figure 3 Screenshot showing the results of running the unit test DealTest.

## Artifacts

### Source Code

The full source code is in a zip archive that was attached along with this document. The archive is titled "*DanielBerryCapstone.zip*".

The code repository is also available at <https://github.com/dberry37388/crm-project>

### Link to Live Version

Because this is a cloud-based application, there is nothing to install on your local device. You can explore the live application by visiting:

To get the full experience, you can register an account of your own, or you can use the login credentials below to access our demo team:

**Username:** jdoe@example.org

**Password:** Wgu2023\*

Please note that when using the sample team and credentials, the application will not be able to send any emails, e.g. password reset.

# Application Maintenance Guide

This section of the document is intended to provide information on setting up your local development environment for maintaining and debugging the application.

## Prerequisites

- A stable internet connection
- Windows, MacOS, or Linux
- Docker
- Node.js v17.2.0 and NPM for compiling assets
- Git (if you plan to use the repository rather than the zipped archive file) or the referenced zip archive
- Ports 3306, 80, 693, 1025, 8900, and 7700 should be free and not in use by any other applications

## Installing Docker Desktop

Our development environment is built using Laravel's built-in Docker solution called sail. This ensures that our development environment remains consistent regardless of which system a developer is using.

Both Windows and macOS make this easy with Docker Desktop

### Installing Docker Desktop on macOS

Instructions for installing Docker Desktop on macOS can be found here:  
<https://docs.docker.com/desktop/mac/install/>.

### Installing on Windows

Instructions for installing Docker Desktop on Windows can be found here:  
<https://docs.docker.com/desktop/windows/install/>

When using a Windows system, please ensure that Windows Subsystem for Linux 2 (WSL2) is installed and enabled. WSL allows you to run Linux binary executables natively on Windows 10. Information on how to install and enable WSL2 can be found within Microsoft's [developer environment documentation](#).

For Windows users, we recommend using [Microsoft's Visual Studio Code](#) editor and their first-party extension for [Remote Development](#).

## Installing on Linux

Instructions for installing Docker Compose on Linux can be found here:

<https://docs.docker.com/compose/install/>

## Using Sail to Start Your Development Environment

Now that Docker is on your machine, we can use Sail to start your development environment. This document will only cover the minimal steps required to boot the development environment. Please visit <https://laravel.com/docs/9.x/sail#introduction> to learn more about Laravel Sail.

*Please note, the recommended path is installing through git and using composer.*

Below are the steps to start the development environment:

### Using the Zip Archive:

1. Unzip the archive to your *project* directory on your local development machine. This can be any directory you plan to work from.
2. From the command line, navigate to the project directory. This is the directory you either unpacked the zip archive or cloned the git repository.
3. Now that you are in the root of the directory, run `./vendor/bin/sail up -d` command to start the sail environment.
4. Generate a new application key by running `./vendor/bin/sail php artisan key:generate`
5. Migrate and seed the database by running `./vendor/bin/sail php artisan migrate --seed`

6. At this point, the application is running, is loaded with test data, and you can visit <http://localhost> in your browser and login with the demo account credentials supplied to view the site.

### Using Git and Composer

If you cloned the application code using the git repository, the additional steps below will be required:

1. From the command line, clone the repository by running the following command: `git clone git@github.com:dberry37388/crm-project.git capstone`
2. Install the package dependencies by running `composer install`
3. From the command line run `cp .env.example .env` to copy the environment variables.
4. Run `./vendor/bin/sail up -d` from the command line to start sail
5. Create an application key by running:  
`./vendor/bin/sail php artisan key:generate`
6. Create the database tables by running the migration and seeders:  
`./vendor/bin/sail php artisan migrate --seed`
7. Build the frontend assets by running `sail npm install && npm run dev` from the command line.
8. At this point, the application is running, is loaded with test data, and you can visit <http://localhost> in your browser and login with the demo account credentials supplied to view the site.

### Having Issues Installing the Development Environment?

If you run into any issues with the development environment, please contact me by sending an email to [dberry8@wgu.edu](mailto:dberry8@wgu.edu).